A Pd framework for the Xth Sense:
enabling computers to sense human kinetic behaviour

Marco DONNARUMMA
Sound Design, ACE, The University of Edinburgh
Alison House, Nicolson Square
Edinburgh, UK, EH8 9DF
m.donnarumma@sms.ed.ac.uk
m@marcodonnarumma.com

Abstract

The Xth Sense is an interactive system for the biophysical generation and control of music. It makes use of muscle sounds[1] produced by a performer as both raw sonic material and control data. Presently the Xth Sense (XS) technology consists of low-cost, wearable biosensors and a Pure Data[2] based application for capture, analysis, real time processing and playback of human muscle sounds. The technical implementation of the XS biosensors has been recently illustrated in [2].

This paper describes the design of the XS software; it is a program that enables a computer to “listen” to the MMG signals transduced by the XS biosensors, to understand the performance main features, and therefore to interact with the performer. After a brief introduction on the nature of the interaction fostered by the XS technology, I focus on the framework main features such as: the XS library, a tabbed dynamic interface (TDI), a MMG features extraction unit, and a graph-on-parent[3] (GOP) routing system for dynamic mapping of gesture to sound.

Keywords

Biophysical music, muscle sounds, modular DSP framework.

1 Computers that sense and act

Interactive music relies on a flowing exchange between a human being and a computer. As Winkler puts it, “nothing is more interactive than a good conversation”; both parties are engaged, they both share ideas and respond to each others inputs. A good conversation is usually open ended, yet bounded by an intangible frame of rules and cultural contingencies, that are - more or less equally - shared by the participants. It is a delicate balance which can be achieved only by setting a “consistent context” that produces “a feeling of mutual understanding without being predictable” [6]. On the other hand, for Rokeby the computer is “objective and disinterested”, and therefore the interaction (e.g. experience) “should be intimate”. Computers are not smart. Man creates their intelligence, Man forges them as interactive agents, and ultimately defines their degree of interaction with the real world. Man can breathe an understanding of human intimacy into the computer “tiny playing fields of integrated circuits” [5].

At the heart of the XS project stands a twofold motivation: to investigate the modalities by which Man's intimate, bodily energy can become digitally tangible; to develop a context of rules and algorithms which would enable computers to *sense* the varied nuances of the body potential, and *act* accordingly. Here, I purposely refer to the term *acting;* I argue that it does not matter how much complex a computing system is, it will always be *pretending* to be able of making sense of human behaviour. Its understanding is an artificial product of Man's choices.

The following sections seek to outline the aesthetic (e.g. the developer's choices) that drives the mutual and creative exchange between the user and the machine within the XS digital framework.

2 Tools of interaction

Whereas the XS biosensors capture the raw, sonic material produced by a performer's muscle contractions, the central brain (e.g. “consistent context”) of the interactive system

1 Technically called mechanical myography (MMG)

2 A programming language for real time signal processing and computer music.

3 A Pd feature that enables a patch or abstraction to have a custom appearance within the calling parent patch.

lies within the XS software. The program leads a computer to develop an understanding of a human kinetic performance by capturing, and analyzing muscle sounds. It also handles the sonic behaviour of a computer by indicating the rules by which it can process and playback in real time the performer's muscle sounds.

The application was developed with Pd-extended 0.42.5 and although it borrows some objects from other libraries[4], it is based on and developed with the XS library, an *ad hoc* collection of Pd-objects. Sections 2.1 outlines its structural characteristic.

2.1 The Xth Sense library

The XS library aims to bring together a set of GOP objects specifically coded to be used with muscle sounds processing, and a discrete amount of general purpose abstractions that facilitate a faster programming. The library is largely inspired by the Pure Data Montreal Abstractions[5] (Pd Mtl). Similarly to the Pd Mtl library, the XS collection aims to provide high-level, standardized objects to accelerate the learning curve of new users, and to ensure a rapid and more rewarding programming environment.

The library includes an overall amount of about a hundred objects. As for the Pd Mtl library, Xth Sense objects are categorized using a pragmatical taxonomy based on function categories; each object is clearly named after his function, and each function category is easily recognizable by means of a self-explanatory prefix. So far ten categories have been implemented: *count*, counters objects; *efx*[6], real time audio processing; *flow*, analysis of numerical data; *gen*, sound generators; *gui*, various GOP tools which can be used to build more complex macros; *mix*, audio mixing; *scale*, scaling of numerical data; *smp*, sample based audio objects; *utils*, general purpose abstractions.

3 The GUI

A relevant feature of the software is its graphical user interface (GUI); this was designed and optimized in order to offer a handy environment for fast performance prototyping, while enabling first-time users to achieve complex operations without dealing with low-level objects. Next, I illustrate each area of the GUI and its function.

3.1 Software anatomy

The software GUI consists of two main canvases: a side panel and a main window. The side panel hosts a browser[7] to search for specific abstractions in a determined folder, a global preset saving system[8], and a bang button which opens the `[pd adc.in]` subpatch[9]; this is the “ear” of the system, it is where the computer captures the muscle sounds. `[pd adc.in]` operates four functions:

1. audio filtering: a bank of band-pass, hi-pass and low-pass filters clear the MMG signal of unwanted frequencies (e.g. noise), while enhancing the amplitude of the structural partials;
2. thresholding: an algorithm measures the root mean square (RMS) of the incoming wave, then, according to a customizable threshold level, either lets the wave enter the next stage or stops it;
3. limiting: the input signal is increased to reach its maximum amplitude avoiding clipping;
4. distribute: a couple of `[send~]/[receive~]` dispatch the sound wave to the main interface, where the processing takes place.

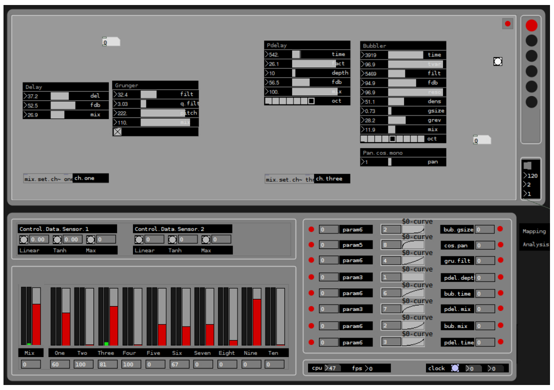

Figure 1: The XS main interface

4 Namely, cyclone, flatspace, iemgui, iemlib, moocow, moonlib. Few objects used come from libraries not included in Pd-extended: iemguts, pdmtl, puremapping and soundhack.

5 See: http://wiki.dataflow.ws/PdMtlAbstractions

6 Later, I refer again to the objects included in the *efx* category; for a better readability I indicate them with the term *efx*.

7 The object is [file.browser_] included in the Pd Mtl abstractions.

8 [presetstore] from the sssad library by Frank Barknecht.

9 A nested patch housed within a parent canvas.

The main canvas includes several macro GOP units; from top to bottom:

- `[pd workspace]`, a wide area in which digital signal processing (DSP) chains can be programmed using the *efx* abstractions[10];
- `[pd control.values]`, a lookup module which displays the numerical values of the features extracted from two XS sensors;
- `[pd analysis]`, it includes the algorithms that compute the muscle sounds to extract meaningful features; the layout is composed of four sliders and two graphical scopes to provide a visual feedback of the control values and the incoming waveform;
- `[pd routing.params]`, a GOP module for one-to-one and one-to-many mapping of MMG based control data to the DSP chains;
- `[pd mixer]`, a 10 channels audio mixer which includes individual, pre-fader aux sends and a master level.

3.2 Usability concerns

The GUI was designed in order to obviate some issues which can be encountered when working with an intricate, multi-task environment in Pd. The first concern was to maintain the readability and usability of complex DSP chains that use GOP abstractions. Usually nesting processes in subpatches is the most common method to avoid Pd programs from being unreadable; however, retrieving, opening and closing several subpatches in a composite framework can sometimes impede the user's creative flow; in fact, such mechanism does not offer an intuitive navigation of the interface which could be beneficial to both first-time users and trained programmers.

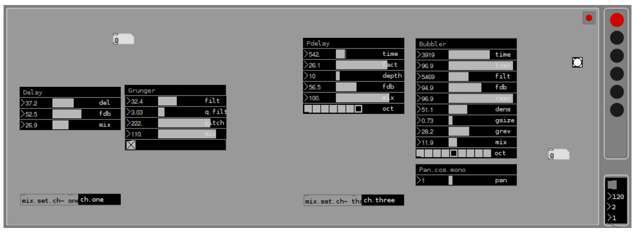

Figure 2: A Pd based TDI

This issue was addressed by the implementation of a TDI, which allows multiple subpatches to be enclosed within a single window, and uses tabs as triggers to switch between sets of subpatches. The TDI used in the XS software adopts animated buttons instead of tabs. The idea of a Pd based TDI was inspired by the concept of "space-awareness" embodied by Zmölnig [7] in the Pd external called `[canvasposition]`[11]. `[canvasposition]` returns "the current position of the object within its containing canvas" and it makes also possible to set a new position. `[canvasposition]` was incorporated in several of the macro GOP units along with a simple function, which sets the position of multiple patches at once. This mechanism enables the user to switch among six `[pd workspace]`, or to toggle the visibility of `[pd analysis]` and `[pd routing-params]` with a single click.

4 Interrelating kinetic behaviour with musical performance

This section describes the aesthetic principles, the design paradigms and the mapping tools that enable the composition of a performance based on the somatic behviour of a performer. How to achieve a seamless, real time interaction with a computer software for DSP, which would guarantee richness of color and sophistication of forms?

The main principles and some technical implementations are illustrated below.

4.1 Performance and design principles

Major aim of designing with MMG audio signals is to avoid a perception of the sound being dissociated from the performer's gesture. The dissociation I point at not only refers to the visual feedback of the performer's actions being disjointed from the sonic experience, but it also, and most importantly, concerns a metaphorical level affecting the listener's interpretation of the sounds generated by the performer's somatic behavior [1]. The use of muscle sounds in this project had to be clearly motivated in order to inform classical approaches to gestural control of music. Therefore, chosen sound processing and data mapping techniques were evaluated according to their capability of enhancing the metaphorical interpretation of the performer's physiological behaviour.

From this perspective, some essential principles were defined as follows:

10 Or any other Pd-object.

11 The object is part of the *iemguts* library by IOhannes Zmölnig, which deals with metaprogramming in Pd.

- to make use of biological sounds as major sonic source and control data;
- to exclude the direct interaction of the performer with a computer and to conceal the latter from the view of the public;
- to demonstrate a distinct, natural and non-linear interaction between kinetic energy and sonic outcome which could be instinctively controlled by the performer;
- to provide a rich, specific and unconventional vocabulary of gesture/sound definitions which can be unambiguously interpreted by the audience;
- to allow the performer to flexibly execute the composition, or even improvise a new one with the same sonic vocabulary;
- to make both performer and public perceive the former's body as a musical instrument and its kinetic energy as an exclusive sound generating force.

4.2 MMG features extraction

Presently, the analysis operations computed by the XS software provide 8 variables; here I describe in detail the features extraction methods.

Since the project deals with sound data (e.g. the muscle sounds), a pitch tracking system may have been a straightforward solution for an automated evaluation and recognition of gestures, however muscle sounds resonant frequency is not affected by any external agent and the pitch seems not to change significantly with different movements [4]. Whereas muscle sounds are mostly short, discrete events with no meaningful pitch change information, the most interesting and unique aspect of their acoustic composition is their extremely rich and fast dynamic; therefore, extraction of useful data can be achieved by RMS amplitude analysis and tracking, contractions onset and gesture pattern recognition. In fact, each human muscle exerts a different amount of kinetic energy when contracting and a computing system can be trained in order to measure and recognize different levels of force, i.e. different gestures.

The subpatch `[pd calibrate]` contains an algorithm that enables the computer to identify which finger is being contracted. This is achieved by adjusting the software sensitivity according to the force threshold of a finger discrete contraction; the threshold is defined by normalizing and measuring the maximum force exertion level of each finger. Although the resemblance between the force amplitude exerted by the minimus (little finger) and the thumb causes the output to be slightly unstable, this method allows the determination of six discrete events (e.g. binary trigger control messages), namely, fingers and wrist contractions.

Secondly, the continuous amplitude average of the performer's arms contractions is measured with `[flow.average]`; next, this value is fed to `[flow.last.max]`, which returns the running maximum amplitude (RMA) of performer's gestures. Often, being that the linear output of this algorithm tightly mirrors the performer's gesture, values can be generated too fast; for this reason, the RMA value is extracted every 2 seconds, then interpolated with the prior one using `[line3]` in order to generate a continuous event; eventually this is normalized to MIDI range and set ready for being mapped.

Lastly, the output of a `[snapshot~]`, which converts the raw MMG signal into a control value, is fed to the `[smooth]` object which applies an equation of single exponential smoothing (SES) in order to forecast a less sensitive continuous control value [3].

4.3 Dynamic data mapping

The composition and design of a performance require concentration, therefore an intuitive mapping system can be very helpful and timesaving. Such system would satisfy a twofold mapping: one-to-one and one-to-many.

In Pd control parameters can be distributed locally in several ways, however the main methods use chord connections created manually or chord-less communication exploiting `[send]` and `[receive]`. I first excluded chord connections for a pragmatical motivation: when rehearsing a piece for the XS the performer's body is completely engaged, thus the use of mouse and keyboard needs to be reduced to the minimum. A chord-less method could not satisfy the requirements of the framework either; de facto, a given Pd-object cannot be reached anyhow, unless the user creates a new `[send]/[receive]` couple and connects them accordingly. Moreover, both approaches imply that Pd is in Edit Mode, which is not always desirable. Such observations motivated an alternative approach. The proposed solution is an enclosed dynamic mapping system, based on a communication protocol which helps the program to identify a given *efx* object and parameter when this is “touched” by the user in Run Mode.

4.3.1 Chordless, dynamic data mapping with [sssad] and [iem_s]

Each *efx* object includes a set of `[sssad]` abstractions that informs the program of its current state (e.g. arguments); in fact, the conventional scope of `[sssad]` is to safely save and restore state data. Nonetheless, it can also be used to identify specific objects in a complex patch. Every time a parameter *foo* of a given object is modified, the embedded `[sssad]` sends to a global receiver a new message, which usually appear as follow:

```
SSSAD foo 125
```

Such message can work very well for a simple patch, but it cannot be used in complex patches as it refers to a global[12] *foo*. Specifically, if more instances of *foo* are used in the same program, all the instances will be updated with the same state. In the case of complex patches that use multiple instances of the same abstraction this issue can be prevented by stating a unique name as an argument for each abstraction. Hence, the new message would read:

```
SSSAD unique-name foo 125
```

Thanks to its TDI, the XS software enables the user to navigate six `[pd workspace]` and to deploy any number of *efx* in order to create DSP chains; arbitrarily naming each *efx* with no defined syntax, but using a unique name can be counterproductive. Therefore, in order to clearly identify each *efx* across the whole interface, the argument of an abstraction can be further elaborated by prepending a path-like syntax: a prefix which indicates the number of the `[pd workspace]` where the object lives (e.g. *wp1*); a single value referring to the object creation order; an acronym which states the class of the object. For instance, a message reporting the *foo* state of the third delay object created in the first workspace would appear as:

```
SSSAD /wp1/3/del.foo 125
```

Such syntax appears more consistent than a unique name, as it includes several information about the object itself, and far more flexible, as it is OSC compatible. But how the system can be aware of where an *efx* lives? In which ways the program can find and communicate with a given *efx*?

This was achieved by implementing a dynamic system based on `[iem_s]`[13]. Every time an *efx* slider is "touched" on the screen by the user, the matching `[sssad]` sends an appropriate OSC message similar to the one described previously. This is intercepted by the macro object `[gui.sssad.send]` which operates four functions:

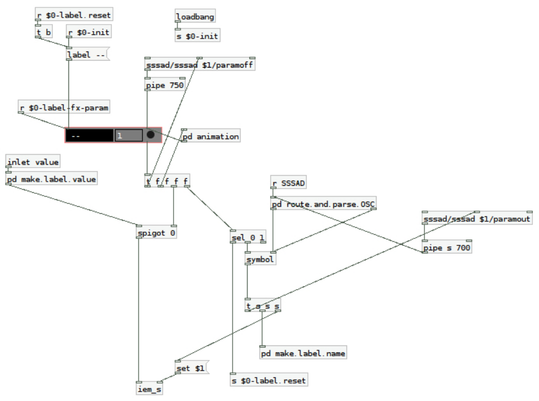

Figure 3: The [gui.sssad.send] abstraction

1. parsing: the messages produced by `[sssad]` are *lists* and contain also the current state of an object, which is needed for preset saving, but not for mapping operations. Thus, first the argument is taken out and secondly, the message is transformed into a *symbol* (for later use with `[iem_s]`);
2. storing: the resulting *symbol* is stored in a `[symbol]` object without being outputted;
3. connecting: when triggered by the user, the stored *symbol* is fed as an argument to `[iem_s]`, which becomes immediately able to send control data to the given *efx* parameter;
4. disconnecting: when triggered by the user, `[iem_s]` is disconnected from the *efx* parameter by reseting its argument to a dummy value.

Unfortunately, the written description of this process may seem too complex, and it does not respect the real world experience of the user. In fact, without considering the computational operations hidden within `[gui.sssad.send]`, all the user has to do can be summed up in four steps:

1. select the source control value to be routed

12 A global variable is intended as a variable which is accessible at all levels of the enclosing context of a program.

13 A settable chord-less sender included in the iemlib library by Thomas Musil.

(by clicking a bang button in `[pd control.values]`);

2. activate one of the 8 inputs within `[pd routing.params]` (it will automatically enable the specified port to receive values from the source selected in step 1);
3. indicate to the system which *efx* parameter the source value needs to be routed to (by "touching" the matching slider);
4. activate one of the 8 outputs of `[gui.sssad.send]` (it will automatically enable the specified port to send values to the parameter selected in step 3).

Additionally, the unit displays matching labels, and provide another GOP abstraction which allows to customize the mapping curve of the source control data, before it is sent to the DSP chain.

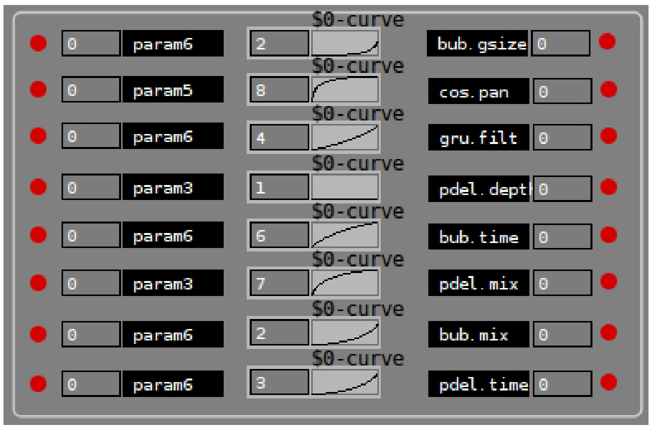

Figure 4: Dynamic routing unit

4.4 Mapping kinetic energy to control data

The implementation of a dynamic mapping system dramatically improved the composition workflow. This paragraph describe some mapping models developed during the composition of *Music for Flesh II*[14], a solo sound piece for the Xth Sense.

A first mapping model deployed the 6 triggers previously described as control messages. These were used to enable the performer to control the real time SSB modulation algorithm by choosing a specific frequency among six different preset frequencies; the performer could select which target frequency to apply according to the contracted finger; therefore, the voluntary contraction of a specific finger would enable the performer to "play" a certain note.

A one-to-many mapping model, instead, used the continuous values obtained through the RMS analysis to control several processing parameters within five DSP chains simultaneously. Being that this paper does not offer enough room to fully describe the whole DSP system which was eventually implemented, I will concentrate on one example which can provide a relevant insight on the chosen mapping methodology; namely, a DSP chain which included a SSB modulation algorithm, a lofi distortion module, a stereo reverb, and a band-pass filter.

The SSB algorithm was employed to increase the original pitch of the raw muscle sounds by 20Hz, thus making it more easily audible. Following an aesthetic choice, the amount of distortion over the source audio signal was subtle and static, thus adding a light granulation to the body of the sound; therefore, the moving global RMS amplitude was mapped to the reverb decay time and to the moving frequency and Quality factor[15] (Q) of the band-pass filter.

The most interesting performance feature of such mapping model consisted of the possibility to control a multi-layered processing of the MMG audio signal by exerting different amounts of kinetic energy. Stronger and wider gestures would generate sharp, higher resonating frequencies coupled with a very short reverb time, whereas weaker and more confined gestures would produce gentle, lower resonances with longer reverb time.

Such direct interaction among the perceived force and spatiality of the gesture and the moving form and color of the sonic outcome happened with very low latency, and seemed to suggest promising further applications in a more complex DSP system.

5 Conclusions

I presented the XS modular framework for real time processing of muscle sounds. When coupled with the XS biosensors this technology appear to disclose promising prospects for an experimental paradigm of musical performance based on MMG.

Along with a technical description of the software and the motivations underpinning its design, the text also addresses some usability issues of the Pd environment. The solutions adopted during the software development were

14 See: http://marcodonnarumma.com/works/music-for-flesh-ii/

15 Narrowness of the filter.

introduced and the abstractions used in this process were unveiled; although these abstractions are at the moment usable only within the XS framework, I believe a further step in the software implementation might be focused on their standardization[16].

Whereas the XS software has been proved fully functional in the composition of *Music for Flesh II*, several improvements to the tracking and mapping techniques can lead to a further enhancement of the expressive potential of the XS framework. Hereafter priority will be given to the extraction of other useful features, to the development of a gesture pattern recognition system and to the implementation of a system for multiple sensors.

References

[1] D. Arfib, J.M. Couturier, L. Kessous, and V. Verfaille: "Strategies of mapping between gesture data and synthesis model parameters using perceptual spaces" Organised Sound, vol. 7, 2003.

[2] M. Donnarumma: "Xth Sense: researching muscle sounds for an experimental paradigm of musical performance" *Proceedings of the Linux Audio Conference* (LAC'11), 2011.

[3] "NIST/SEMATECH e-Handbook of Statistical Methods." Single Exponential Smoothing, e-Handbook of Statistical Methods, 2003. Available at: http://www.itl.nist.gov/div898/handbook/pmc/section4/pmc431.htm [Accessed July 10, 2011].

[4] G. Oster, and J. Jaffe: "Low frequency sounds from sustained contraction of human skeletal muscle" Biophysical Journal, vol. 30, Apr. 1980, pp. 119-127.

[5] D. Rokeby: "Very Nervous System". Available at: http://homepage.mac.com/davidrokeby/vns.html [Accessed July 10, 2011].

[6] T. Winkler: *Composing Interactive Music: Techniques and Ideas Using Max*. The MIT Press, 2001.

[7] I.M. Zmölnig: "Reflection in Pure Data" *Proceedings of the Linux Audio Conference* (LAC'09), 2009.

16 In order to allow the larger Pd community to deploy these abstractions outside of the XS software.